Funded by the Horizon 2020 Framework Programme of the European Union

D3.2
AsTeRICS Plugin Set

Easy Reading

PROJECT DOCUMENTATION SHEET	
Project Acronym	Easy Reading
Project Full Title	Easy Reading
Grant Agreement	780529
Call Identifier	H2020-ICT-2016-2017
Topic	ICT-23-2017
Funding Scheme	RIA (Research and Innovation action)
Project Duration	30 months (January 2018 – June 2020)
Project Officer	Michael Busch, European Commission Directorate-General for Communications Networks, Content and Technology, Unit &-3, L-2557 Luxembourg, +352.4301.38082
Coordinator	Universität Linz (JKU), Austria
Consortium partners	Kompetenznetzwerk (KI-I), Austria Technische Universität Dortmund (TUDO), Germany In der Gemeinde Leben gGmbH (IGL), Germany FUNKA Nu AB (FUNKA), Sweden Texthelp Ltd (TEXTHELP), United Kingdom VÄSTRA GÖTALANDS LÄNS LANDSTING (DART), Sweden GEIE ERCIM (ERCIM), France AHTENA I.C.T. Ltd (ATH), Israel
Website	https://www.easyreading.eu

DELIVERABLE DOCUMENTATION SHEET	
Number	Deliverable D3.2
Title	AsTeRICS Plugin Set
Related WP	WP3
Related Task	T3.2
Lead Beneficiary	KI-I
Author(s)	Barbara Wakolbinger, Stefan Parker
Contributor(s)	Gerhard Nussbaum
Reviewers	Alan McCaig
Nature	Other
Dissemination level	Public
Due Date	30/06/2019
Submission date	25/06/2019
Status	Final Version

QUALITY CONTROL ASSESSMENT SHEET			
Issue	Date	Comment	Author
1	17/06/19	Initial review	Alan McCaig
2	19/06/19	Final review	Alan McCaig

ACKNOWLEDGEMENT

This document is a deliverable of the Easy Reading project, which has received funding from the European Union’s Horizon 2020 Programme for Information and Communication Technologies under Grant Agreement (GA) Nb #780529.

Executive Summary

D3.2 describes a set of needed additional plugins for the AsTeRICS system, which will help to develop an AsTeRICS model that can draw conclusions on a user's cognitive stress level, based on various sensor readings.

The HRVRmssdFromRR plugin calculates Heart Rate Variation Root Mean Square of Successive Differences, which can indicate cognitive stress. The BlinkChangeDetector takes the eye's current state (open or closed) as input and calculates changes in blink duration and rate, both of which tend to change with cognitive load. Apart from these data acquisition and processing plugins, the other described plugins are "helper-plugins" needed for reasoner development and user data collection for development. The StringExtractor extracts a substring from a given input string based on start and end delimiter strings. The TimestampWriter stores the time that has passed since the start of the model or the last reset. Outputs are based on the local system's time zone, which is also the base of the DateToDouble plugin, which converts a date string to a Unix epoch timestamp. Finally, the JsonParser is also a helper plugin that is needed because the communication within the Easy Reading system will be realised through passing JSON objects.

The present document is a deliverable of the Easy Reading project which is funded by the European Union's Horizon 2020 Programme und Grant Agreement #780529.

Table of Content

Introduction

User tracking in the Easy Reading project is supposed to track the user's cognitive stress at the moment of viewing a document. For this purpose, it is necessary to gather and process sensor readings in order to draw conclusions on the user's current cognitive state. The AsTeRICS system (www.asterics.eu) provides a massively useful tool for this, since developers can resort to a pool of around 200 plugins for sensors, actuators and data processing. Any functionality not yet supported by the framework, can easily be added by writing new plugins for the system, which then can be used in sophisticated models together with all the pre-existent plugins. For data collection and processing in the scope of the Easy Reading project, a number of additional plugins are needed, which are described in this deliverable. Every plugin can have data ports for input and output as well as event listeners, event triggers and a number of properties, which are also described in detail below. Finally, the prospective use in Easy Reading is explained for every plugin, thus showing why it is needed.

Descriptions of new Plugins

In the following, the newly implemented plugins are described in detail, whereat the title of each section represents the plugin name as shown in the ACS (AsteRICS Configuration Suite). All port names, event triggers, event listeners and properties are named equal in this document as in the ACS as well.

HRVRmssdFromRR

Calculates and outputs the Heart Rate Variability (HRV) in terms of RMSSD (Root Mean Square of Successive Differences) in milliseconds (ms) based on incoming R-R intervals (also in ms).

A sliding window of the x most recent R-R intervals, x being the property rmssdWindowSize, is used for the calculation, which is first started as soon as there have been x samples since the last reset (or model start).

Calculation and collection of samples can be paused and continued via event listener ports. The output port rmssd only delivers a value, if there is an actual calculation (i.e. currently no pause and enough samples).

Output Ports

rmssd (double): Provides the calculated Root Mean Square of Successive Differences in ms, calculated from the collected input port rrInterval’s signals.

Input Ports

rrInterval (double): This port must deliver the R-R interval in ms, i.e. the time interval between the most significant, the highest, peaks (the R-peaks) of two consecutive QRS' of an ECG (electrocardiogram).

Event Listeners

resetCalculation: When the event is detected, the plugin discards previous R-R intervals and resets the counter of intervals used for the calculation. This does not influence running or paused states, i.e. if the plugin is currently paused, it will stay paused, otherwise there is a recalculation as soon as there are (again) enough samples.

pauseCalculation: When the event is detected, from now on RMSSD is no longer recalculated and no value is sent to the output port, but R-R values from the input port continue to be stored (more recent ones overwrite existing ones due to the sliding window). This can be used in order to wait for more meaningful R-R samples before the next calculation, respectively further outputs.

continueCalculation: The event must be fired after each pauseCalculation in order to continue RMSSD calculation and sending the result to the output port again.

pauseComponent: Completely pauses the component's activity, i.e. from now on no RMSSD is recalculated and sent to the output port. In addition to a pauseCalculation, no more R-R values from the input port are collected, either.

continueComponent: To be used after pauseComponent in order to collect R-R values from the input port again (values aren’t reset but progressively overwritten due to the sliding window) and to continue RMSSD calculation and sending the result to the output port.

Event Triggers

rmssdRecalculated: Triggers whenever an RMSSD calculation has been finished and the current RMSSD value is available at the output port, i.e. if rmssdWindowSize (property) is 100, it will trigger first when 100 intervals were received and calculation has finished, then after each further interval and calculation completion.

Properties

rmssdWindowSize (integer, default: 100): The number of R-R intervals that are used for each RMSSD calculation, thus the sliding window size. Example: If this is set to 100, the 100 most recent R-R intervals are taken into consideration and calculation is not started before at least 100 values have been received at the input port (since the start or a possible reset). A valid value must be > 1, otherwise it is replaced by the default value.

Prospective use in Easy Reading

RMSSD was found to be a suitable indicator for a user being stressed, in terms of heart rate measurement and its analysis in the time domain. Thus this is one of the plugins that shall provide pre-processed input and clues for the reasoner implemented with AsTeRICS.

BlinkChangeDetector

The plugin detects blinks, based on the eyes' open and closed states which are indicated by events, which it listens to, and then calculates and outputs blink-related metrics. This plugin does not detect the eyes' states itself! Though, in order not to bother the plugin user (the AsTeRICS model developer) with blink detection, it is sufficient to externally fire events of both eyes' state currently being open or closed (no matter whether the state has changed or stayed the same for a while). By the plugin listening to the events eyesAreOrBecomeOpen and eyesAreOrBecomeClosed is also how the model developer can individually specify a temporal granularity for blink metrics output, e.g. by external timers and thresholds, even if the eyes' state has not changed.

Eyes state: open or closed, always with reference to both eyes simultaneously.

Basic Terminology for Blink States: A blink's start is defined as both eyes' state having simultaneously changed from open to closed, the blink's end is indicated by both eyes' being reopened again. There is an ongoing (a "current") blink as soon as a blink's start is detected.

Provided Metrics: The respective "forwarding" event listeners allow recalculating and forwarding blink rate or blink duration to the output ports on demand. There is a recalculation before every output.

Further, blink rate and blink duration are recalculated and sent to the output ports when the eyes' state changes, and, if specified via properties, in a certain interval. If the eyes' state events are detected but the state has not changed, there is only a recalculation and output when propOutputIntervalMs is set and has passed by since the last state-based output.

Implications: Note, that the duration changes while the eyes are closed and is set to 0 milliseconds when the eyes' state changes to closed. It is longest at the first eyesAreOrBecomeOpen event in a row and then set back to 0 milliseconds again till the next eyesAreOrBecomeClosed is detected.

In contrast, the rate changes steadily and is unlikely to ever be 0 Hz again, independently from eyes being closed or open.

A blink is included in the rate calculation as soon as it has started (i.e. the eyes changed to closed).

Trend Detection: Changes in blink rate and blink duration are analysed at certain eyes' states respectively state changes. Then the plugin fires events, if certain thresholds are exceeded respectively undercut.

The detectable trends (rate increase/decrease and duration prolongation/shortening) work with the same basic detection algorithm, described below, but using trend-specific properties (see Properties Section).

Trend Detection Triggers in detail (also see Event Listeners Section): The rate trend is analysed, when a blink starts or the output interval leads to recalculation. The duration trend is only analysed, when a blink ends (as then the final duration is clear and as there cannot be a duration analysis while no blink is going on, either). Trend detection is never done when there is a rate or duration recalculation and output due to forwarding events.

Basic Trend Detection Algorithm

Blink duration trend detection as well as blink rate trend detection allow certain thresholds. All of them are optional and specify whether only an increasing, only a decreasing, both or no trends are detectable.

Each calculated value not undercutting the lower and not exceeding the upper threshold is "within range".

If the most recently calculated value is passed on to trend detection, three cases are possible:

- The value exceeds the upper threshold which starts the detection algorithm for a currently ongoing increasing trend.
- The value undercuts the lower threshold which starts the detection algorithm for a currently ongoing decreasing trend.
- The value is within range which means an ongoing trend is not interrupted but no new trend can be detected either.

If one of the thresholds is not set, the corresponding case cannot appear but does not influence the others.

If trend detection into one direction is initiated, an opposite trend is always interrupted.

Whenever starting trend detection into one direction, the corresponding internal counter (above upper threshold or below lower threshold) is set to 1, then the buffer of earlier calculated values is walked through backwards, i.e. the more recent values first. Each value within the range does not affect trend detection, whereas each crossing a threshold in the same direction increases the counter and in the opposite direction interrupts this turn of trend detection.

The algorithm terminates with firing the corresponding trend detection event, as soon as the counter exceeds howManyDurationOutliers (for a duration trend) or howManyRateOutliers (for a rate trend), respectively.

The algorithm terminates with not firing any detection event, if the start of the buffer was reached or if it was interrupted by a value crossing the opposite threshold.

Output Ports

blinkDurationMs: Outputs the currently recalculated duration in milliseconds (ms) since blink start, whenever the event forwardCurrentBlinkDuration is detected, eyes' state changed to closed, or eyesAreOrBecomeOpen or eyesAreOrBecomeClosed are detected and propOutputIntervalMs passed by. Outputs 0, if currently the eyes are open (no current blink – indicated by the event eyesAreOrBecomeOpened).

blinkRateHz: Outputs the current recalculated blink rate (in Hertz) since the plugin's start, whenever the event forwardCurrentBlinkRate is detected, or when eyesAreOrBecomeOpen or eyesAreOrBecomeClosed are detected and propOutputIntervalMs passed by.

It is possible to make the plugin calculate it for a specified observation period, rather than since model start, this can be set by the property rateObservationPeriodMinutes.

Input Ports

None.

Event Listeners

eyesAreOrBecomeClosed: The event represents the closed-state of the eyes and the listener handles, whether it indicates that a new blink has just started or eyes have not even been opened in the meantime (no previous eyesAreOrBecomeOpen since last firing this event). This listener is responsible for triggering blinkStarts.

Considering propOutputIntervalMs and the current blink state, it recalculates blink duration, blink rate and initiates trend detection for the blink rate.

eyesAreOrBecomeOpen: The event represents the open-state of the eyes and the listener handles, whether it indicates that a blink has just ended or eyes have not even been closed before (no previous eyesAreOrBecomeClosed since last firing this event). This listener is responsible for triggering blinkEnds.

Considering propOutputIntervalMs and the current blink state, it recalculates blink duration, blink rate and initiates trend detection for the blink rate and blink duration.

forwardCurrentBlinkDuration: Sends the time that has passed by since blink start to the output port. 0, if currently not blinking (i.e. if eyes are open). It does not do any threshold evaluations (trend detection) or trigger events!

forwardCurrentBlinkRate: Sends the recalculated blink rate, no matter whether there is currently a blink (i.e. eyes closed) going on. A possible ongoing blink is already included in the rate calculation. It does not do any threshold evaluations or trigger events!

Event Triggers

blinkRateIncreased: Triggers when the detection of an increasing blink rate trend was positive.

Also, the properties rateObservationPeriodMinutes and trendsBufferSize are considered and must therefore be set meaningfully (or be disabled).

blinkRateDecreased: Triggers when the detection of a decreasing blink rate trend was positive.

Also, the properties rateObservationPeriodMinutes and trendsBufferSize are considered and must therefore be set meaningfully (or be disabled).

blinkDurationLonger: Triggers when the detection of an increasing blink duration trend was positive.

Also, the property trendsBufferSize is considered and must therefore be set meaningfully (or be disabled).

blinkDurationShorter: Triggers when the detection of a decreasing blink duration trend was positive.

Also, the property trendsBufferSize is considered and must therefore be set meaningfully (or be disabled).

blinkStarts: Triggers when a blink's start is detected during the handling of the eyesAreOrBecomeClosed event.

blinkEnds: Triggers when a blink's end is detected during the handling of the eyesAreOrBecomeOpen event.

Properties

outputIntervalMs (integer, default: -1): Defines how often the blink rate and duration (also 0, if currently not blinking) are sent to the output ports without the forwarding events being fired. This is only relevant, if the eyes' state has not changed as at blink end or blink start the output ports always receive data. This property was introduced in order not to send values too often (e.g. in a 1ms-interval, if the source for eyes state detection such as a camera plugin would fire events that often) but to adapt the outputs for visualization in an oscilloscope etc. The unit for the interval is milliseconds (ms).

Rate is always recalculated and sent to the output port when a blink starts, the duration, if a blink starts (0 ms) or ends. If those outputs, together with output (recalculation and) forwarding events on demand is sufficient, and no continuous output is desired, it can be disabled by setting 0 or a negative value.

It is not possible to turn off the outputs, just described for blink starts or ends.

Used for blink rate trend and blink duration trend detection

trendsBufferSize (integer, default: 10): Defines how many blink rates or blink durations can be looked back for detecting blink rate and blink duration outliers in trend detection.

This buffer must not be lower than howManyRateOutliers and howManyDurationOutliers properties, as then the conditions to fire the corresponding events can never be fulfilled! Value 0 disables the buffer (only meaningful if the needed number of outliers is 1 or disabled). Negative values are invalid and the default value is used. This must not be confused with rateObservationPeriodMinutes which restricts the period for which blinks are considered when calculating the current blink rate.

Used for blink duration trend detection only

threshDurationHighMs (integer, default: 500): Defines the blink duration in milliseconds (ms) above which a blink is detected as an outlier, representing a prolonged blink duration. This upper threshold can be disabled by setting it to 0 or a negative value.

threshDurationLowMs (integer, default: -1): Defines the blink duration in milliseconds below which a blink is detected as an outlier, representing a shortened blink duration. This lower threshold can be disabled by setting it to 0 or a negative value.

howManyDurationOutliers (integer, default: 3): Defines how many blink duration outliers must occur in one trend direction within the buffered durations for a duration trend to be detected. If a trend shall be detected, as soon as there is one outlier, it must be set to 1, duration trend detection can be disabled by setting it to 0 or a negative value. The 1-outlier option especially makes sense when no buffer is used.

Used for blink rate trend detection only

threshRateLowHz (double, default: -1): Defines below which rate in Hertz (Hz) a blink rate must be, in order to count towards a decreasing blink rate trend. This lower threshold can be disabled by setting it to 0 or a negative value.

threshRateHighHz (double, default: 1): Defines above which rate in Hertz (Hz) a blink rate must be, in order to count towards an increasing blink rate trend. This higher threshold can be disabled by setting it to 0 or a negative value.

howManyRateOutliers (integer, default: 3): Defines how many blink rate outliers must occur in one trend direction within the buffered rates for a rate trend to be detected. If a trend shall be detected, as soon as there is one outlier, it must be set to 1, rate trend detection can be disabled by setting it to 0 or a negative value. The 1-outlier option especially makes sense when no buffer is used.

rateObservationPeriodMinutes (integer, default: 2): Defines the length of the observation period for the blink rate in minutes, e.g. if 3, the number of blinks during the last 3 minutes (sliding window) is considered when calculating the blink rate for this 3-minutes-period.

If this property is used, no memory control is done, thus it is recommended not to use a period longer than 1 day (1440 minutes). If all blinks and the whole time period since model start (memory control restricts it to several hours though) shall be considered, the observation period can be disabled by setting it to 0 or a negative number. Note, that the rate for a natural blinking behavior gets steadily flatter then, as time passes by and might not be representative for trend detection.

Prospective use in Easy Reading

This plugin already does a little preparatory work for stress detection (in terms of cognitive load analysis) to help the reasoner, later being implemented with AsTeRICS.

It was found that changes in blink rate and blink duration are significant indicators for the demand of cognitive load and thus mental stress level. The plugin can get information on eyes' states (open, closed) from different sources, such as a cheap standard webcam via the AsTeRICS FacetrackerCLM plugin. The fact that it only needs the eyes' state and no previous blink detection or metrics such as timing measurements but does it itself, makes it highly reusable for different eye tracker sources, i.e. sensor and other processor plugins.

The options of providing the calculated metrics at specified output ports on demand but also in regular intervals enables visualization, e.g. with the Oscilloscope plugin, which helps with interpretation.

It will be a main part of the AsTeRICS reasoner to find and utilize suitable threshold rates and durations that indicate cognitive load or stress level and further to combine them. This research part and the later implementation can be perfectly done by the huge amount of properties, interacting with each other.

StringExtractor

Extracts a subtext from a given input text and forwards it to the output port as soon as extraction was done based on start and end delimiter strings.

Delimiters must be in the correct order and non-overlapping. In case of several occurrences, always the first occurrence of the start and the end delimiter are used. The event trigger also fires during successful extraction, which is also the case for an empty extracted text (also handed to the output port).

Output Ports

extractedText (string): Represents the subtext extracted from the input text based on delimiter specifications, i.e. the text between start and end delimiter (can also be an empty string).

Input Ports

inText (string): Provides the input text, from which a subtext shall be extracted and sent to the output port, based on start and end delimiter specifications.

Event Listeners

None. (Each new inText is checked for the existence of start and end delimiter.)

Event Triggers

textExtracted: Triggers when the end delimiter was detected after the start delimiter (not, if the order is wrong or one is missing or overlapping, i.e. within the other) and thus a subtext was extracted and sent to the output port. (An empty string is a valid output that also triggers!)

Properties

startDelimiter (string, default: START): Defines the start string between which and the end string the text is extracted (no regular expression!)

endDelimiter (string, default: END): Defines the end string between which and the start string the text is extracted (no regular expression!)

Prospective use in Easy Reading

This is a very important helper plugin for the Easy Reading AsTeRICS model. Tagged/annotated texts received via the SerialPort plugin, i.e. from a COM port as the receiver in a Bluetooth communication, can be easily processed to meet the requirements to serve as inputs for other plugins such as HRVRmssdFromRR (maybe after further data/type conversion steps).

Also tagged error messages from Bluetooth communication can be easily detected, extracted and (if desired) shown to the Easy Reading users.

Finally, different Bluetooth sources can be annotated and with the help of this plugin their data can be automatically forwarded to the desired plugins or along certain model paths accordingly.

TimestampWriter

This plugin stores the time in milliseconds (ms) that passed by since the last reset (or start). Outputs are the (via date string) formatted and numerical milliseconds and also a formatted absolute UNIX epoch timestamp plus the absolute UNIX timestamp in ms. The time zone of the local system is used!

Output Ports

timePassedMs (double): Milliseconds (ms) since model start or last component reset.

timePassedFormatted (string): Milliseconds since model start or last component reset - formatted as date string.

timestampUnixMs (double): Unix timestamp in milliseconds (ms).

timestampUnixFormatted (string): Unix timestamp – absolute milliseconds formatted as date string.

Input Ports

None. Milliseconds are either absolute Unix epoch timestamps or relative intervals between events.

Event Listeners

resetStartTimestamp: Resets the start timestamp for calculating the time passed by (the first timestamp is set at model start).

sendOutputs: Formats and sends outputs, triggers the event timestampFormatted when outputs are available at the ports.

Event Triggers

timestampFormatted: Triggers when output values (formatted and as milliseconds) are available at the output ports.

Properties

timestampFormat (string, default: “dd.MM.yyyy-HH:mm:ss.SSS”): Format that must be valid according to java.text.SimpleDateFormat. It is used for formatting the UNIX timestamp as a date string. If it is invalid or null, the default date format is used.

diffTimeFormat (string, default: “HH:mm:ss.SSS”): Format that must be valid according to java.text.SimpleDateFormat but for meaningfulness with only time components. It is used for formatting passed milliseconds (since start/reset) as a date string. If invalid or null, the default date format is used.

Important Usage Note: Only the format for time components shall be set, as otherwise the Unix epoch start date components (01.01.1970 at 00:00:00.000 o’clock) are added. e.g. if 5 seconds have passed by, using date components the output would look like 01.01.1970-00:00:05.000!

Prospective use in Easy Reading

The plugin is needed for aligning data from different tracking sources (heart rate tracker, eye tracker or webcam) to each other (i.e., “synchronizing” them). This is necessary as most of the sensor or communication plugins only provide raw data without a temporal reference, which the later implemented (AsTeRICS) reasoner could likely not make use of, once data was just sequentially written into text or csv files.

Usefulness and recoverability of earlier tracked data from user tests (during the early research phases on generally useful metrics) for later phases such as reasoner implementation would not be given without also storing the timestamps.

While mathematical methods and visualizations are likely to demand numerical milliseconds-based timestamps (maybe even relative ones, with regard to the model start), readability of text-based files will improve when (formatted) absolute date and time are printed.

DateToDouble

Converts a date string to a Unix epoch timestamp in milliseconds, using the default time zone and a property-defined date format.

Output Ports

timestampUnixMs (double): The converted Unix epoch timestamp in milliseconds.

Input Ports

dateString (string): Date string input to be converted using the date format given by the property. The event formatInput is also fired internally whenever this input port receives data.

Event Listeners

formatInput: Initiates conversion for the current input date string, triggers dateFormatted when done.

Event Triggers

dateFormatted: Triggers when an output value is available at the output port.

conversionFailed: Triggers when a conversion attempt failed (might happen if inputs are missing or do not fit the specified date format).

Properties

dateFormat (string, default: “dd.MM.yyyy-HH:mm:ss.SSS”): The format of the incoming date string. It is used for conversion into a UNIX timestamp in milliseconds. If invalid (according to java.text.SimpleDateFormat) or null, the default date format is tried. If conversion fails because of missing or invalid input conversionFailed is triggered.

Prospective use in Easy Reading

Rather than after the release, this helper plugin is mainly important for the phases of AsTeRICS model creation and reasoner implementation as it might be the case that in some past user tracking tests (early research phase) data were persisted with formatted timestamps but are then needed with numerical milliseconds timestamps for visualizations, alignment with other sources’ tracking data or mathematical methods.

Without this plugin user tracking tests would need to be redone for some cases which is of course not representative, if even possible (i.e. it’s not meaningful to let the same users do the same tasks again as the outcome won’t be the same in a second turn; there won’t be enough time capacities for redoing several test sessions).

JsonParser

Receives a JSON (JavaScript Object Notation) string and JSON field name as input and parses it into a JSON object to send one or more field values (several ones, only if the keepParsedObject is true and other jsonFieldName input values follow).

Output Ports

jsonFieldValue (string): The value of the JSON field, identified by jsonFieldName, converted to a string (also "" is valid; an explicit null is converted to "null"). In case of failure, it will be "" and the error port provides details.

latestReadFieldName (string): The name of the field whenever an attempt of reading a field value (from the parsed JSON object) has been completed. This output can only be "" if the field name was ""/null/unset, in which case the error output port provides details.

error (string): Receives a value (different from "") whenever an attempt of input parsing or reading the next field fails. No matter whether it is because of a missing JSON object (as needed for reading a field but not previously parsed) or missing or invalid input port values. It receives "" (= no error) at the next successful parse or read. Thus it can be used for showing error messages.

Input Ports

jsonInputString (string): The JSON-formatted string to be converted into a JSON object.

jsonFieldName (string): The key/name of the JSON object's field, of which the value shall be sent to the output port (most recent name from this port is used, “” causes an error at readNextField event).

keepParsedObject (boolean, default: true): Optional. If connected, it overwrites the keepParsedObject property.

Event Listeners

For details on which errors are handled and error messages are provided for, see description of the error output port and of the event triggers that cause an error port output.

parseInput: Parses the most recently received value from the input port jsonInputString into a JSON object - from which later the desired JSON field (identified by the jsonFieldName input) value is read and sent to the jsonFieldValue output port.

readNextField: Triggers the next attempt of reading the value of the currently desired JSON field (identified by jsonFieldName) from an already parsed JSON object.

This event must be used after parseInput (which must be repeated if keepParsedObject is set to false). parseInput, in combination with a well-matched keepParsedObject property/port, has to assure that there is already a JSON object from which values can be read when firing readNextField.

rejectParsedObject: Unsets the parsed JSON object for the case that keepParsedObject is true (via property or overwritten by input port value) or if it is false but no successful reading of any JSON field has happened (see property description).

Event Triggers

inputParsed: Triggers to indicate that the plugin is done with parsing the JSON input (string) into a JSON object.

fieldValueAvailable: Triggers when the value of the desired JSON field (specified by the input jsonFieldName and now identified by the output latestReadFieldName) is available at the jsonFieldValue output port.

Event Triggers that cause an error port output

fieldValueNotFound: Triggers when trying to read a field but when there is no field with the name given at the input port within the parsed JSON object (string).

fieldValueInvalidFormat: Triggers when trying to read a field but when there has been an error returning the value of the desired JSON field. The value has a wrong/an invalid format but is found (i.e. the key is existent). This is the case whenever a found value cannot be parsed to a string (which is the output format at the output port jsonFieldValue).

invalidInputFormat: Triggers when trying to parse the JSON input string but when it cannot be parsed into a JSON object (to later read values from).

missingJsonInput: When trying to parse the JSON input string but when there has been no value at the input port jsonInputString.

missingFieldName: Triggers when trying to read from an already parsed JSON object but when there has been no value at the input port jsonFieldName.

Properties

keepParsedObject (boolean, default: true): Specifies, whether to keep the parsed JSON object (the one resulting from the latest parsing of jsonInputString) until the next value appears at the jsonInputString port – to read further fields in the meanwhile.

Usage Note: If a field reading attempt fails, a JSON object that has potentially been stored will not be unset, independently from the value of this property. This is to assure that each JSON input delivers at least one field value or otherwise is replaced by the next JSON input. In order to force the parsed JSON object's rejection, rejectParsedObject must be triggered.

Prospective use in Easy Reading

The aim of this plugin is to use it for utilizing web page interaction information or profile information, provided by the browser plugin (both to be implemented in close cooperation with other Easy Reading partners). The JsonParser plugin is not in use at the current research and implementation state but will likely play an important role when implementing the (AsTeRICS) reasoner as it helps customizing the reasoner, depending on information that can not be received from (only) tracking via the hardware sensors.

Conclusions

While some of the plugins described above already deliver concrete clues (like the RMSSD) or already try to draw conclusions ready for use in reasoning (like blink rate or duration trend detection), others are “helper-plugins” needed for different purposes and will make sense only in the context of the reasoner model. Important is to note that reasoner development will start now and is anticipated to be a complex task that might require additional plugins that could not be anticipated yet and will have to be written on the way, if deemed necessary. Also adaptations and optimisations of presently available plugins might become necessary, as development progresses.